ECM3408 Enterprise Computing
Laboratory 1

D. Wakeling

This laboratory develops a first microservice that manages a database of airport codes and names. This microservice implements (some of) the RESTful *Store* archetype. See Figure 1.

Code	Name
LAX	Los Angeles
LHR	London Heathrow
MXP	Milan Malpensa
PEK	Beijing

Table 1: A table of airport codes and names.

Activity 1

Use the Windows 11 search to get an Anaconda command prompt. Create a directory to do your work, and change to that directory.

```
mkdir laboratory1
cd laboratory1
```

Activity 2

The SQL database implementation that we shall use is SQLite, a popular and easy-to-use relational database used by many enterprises. Use a text editor to enter the following code for a `database.py` module.

```
import repository

db = repository.Repository("airports")
```

The *Repository* design pattern allows a program to work with a database without becoming mired in detail. This pattern in turn uses the *Object-Relational Mapping* (*ORM*) design pattern, which allows data to be mapped between an object fields and a relational database rows. Use a text editor to save the following code for a repository module in the file `repository.py`.

```
import sqlite3

class Repository:
  def __init__(self,table):
    self.table = table
    self.database = self.table + ".db"
    self.make()

  def make(self):
    with sqlite3.connect(self.database) as connection:
      cursor = connection.cursor()
      cursor.execute(
        f"CREATE TABLE IF NOT EXISTS {self.table} " +
        "(code TEXT PRIMARY KEY, name TEXT)"
      )
      connection.commit()

  def clear(self):
    with sqlite3.connect(self.database) as connection:
      cursor = connection.cursor()
      cursor.execute(
        f"DELETE FROM {self.table}"
      )
      connection.commit()

  def insert(self,js):
    with sqlite3.connect(self.database) as connection:
      cursor = connection.cursor()
      cursor.execute(
        f"INSERT INTO {self.table} (code,name) VALUES (?,?)",
        (js["code"],js["name"])
      )
      connection.commit()
      return cursor.rowcount

  def update(self,js):
    with sqlite3.connect(self.database) as connection:
      cursor = connection.cursor()
      cursor.execute(
        f"UPDATE {self.table} SET name=? WHERE code=?",
        (js["name"],js["code"])
      )
```

```
      connection.commit()
      return cursor.rowcount

  def lookup(self,code):
    with sqlite3.connect(self.database) as connection:
      cursor = connection.cursor()
      cursor.execute(
        f"SELECT code, name FROM {self.table} WHERE code=?",
        (code,)
      )
      row = cursor.fetchone()
      if row:
        return {"code":row[0],"name":row[1]}
      else:
        return None
```

Activity 3

A microservice manages a database of airport codes and names. Use a text editor to enter the following code for an Airports microservice in the file `airports.py`.

```
import database
from flask import Flask, request

app  = Flask(__name__)

@app.route("/airports/<string:code>",methods=["PUT"])
def endpoint1(code):
  js    = request.get_json()
  code2 = js["code"]
  name  = js["name"]
  if code2 != None and name != None and code == code2:
    airport = {"code":code,"name":name}
    if database.db.lookup(code) != None:
      if database.db.update(js):
        return "",204 # No Content
      else:
        return "",500 # Internal Server Error
    else:
      if database.db.insert(js):
        return "",201 # Created
      else:
        return "",500 # Internal Server Error
  else:
    return "",400 # Bad Request
```

```
if __name__ == "__main__":
  app.run(host="localhost",port=3000)
```

Activity 4

Start the Airports microservice with the command

```
python airports.py
```

Activity 5

Some tests confirm that the Airports microservice is working. Use a text editor to save the following code for test script in the file `test-airports.py`.

```
import requests
import unittest
import database

airports = "http://localhost:3000/airports"

class Testing(unittest.TestCase):
  ############################################################
  ## Test [1]                                               ##
  ############################################################
  def test1(self):
    database.db.clear()
    code = "LAX"
    name = "Los Angeles"

    hdrs = {"Content-Type":"application/json"}
    js   = {"code":code,"name":name}
    rsp  = requests.put(f'{airports}/{code}',headers=hdrs,
             json=js)

    self.assertEqual(rsp.status_code,201)
```

Activity 6

At another Anaconda command prompt, run the test script with the command

```
python -m unittest test-airports.py
```

Activity 7

Add some code to your Airports microservice module to read an airport record from the database, given its code.